Proceedings of the Japan-USA Symposium on Flexible Automation, Ann Arbor, MI , July 23 - 26, 2000.

AN OPEN ARCHITECTURE INSPECTION SYSTEM

Elena R. Messina Hui-Min Huang Harry A. Scott

National Institute of Standards and Technology
Intelligent Systems Division
100 Bureau Drive Stop 8230
Gaithersburg, MD 20899-8230
{messina,huang,scott}@cme.nist.gov

ABSTRACT

The benefits of open architectures are not currently available for most manufacturing equipment. We describe a testbed for experimenting with an open architecture controller for an inspection system. An initial taxonomy of categories of openness is proposed, culminating in "plug and play," which does not require a software rebuild. The testbed and its constituent components are described with respect to these categories of openness. We conclude that our testbed demonstrates all of the openness categories except plug and play.

INTRODUCTION

The machine tool industry is evolving towards the pervasive use of open architecture controls, many of which are available separately from the "iron." The benefits of open architecture controllers include flexibility, customizability, lower costs, and upgradeability. Manufacturing equipment other than machine tools has not necessarily come as far in availability or widespread use of open architecture controls. In this category are robotic systems and inspection systems, such as coordinate measurement machines. At the National Institute of Standards and Technology's Intelligent Systems Division, we are developing open architecture controllers for a variety of manufacturing applications. In this paper, we describe our controller for an integrated inspection system.

NOMENCLATURE

BG	Behavior Generation
CMM	Coordinate Measuring Machine
DMIS	Dimensional Measuring Interface Standard
EX	Executor
FBICS	Feature-Based Inspection and Control
IEEE	Institute of Electrical and Electronics Engineers
ISAM	Intelligent System Architecture for Manufacturing
KB	Knowledge Base
NML	Neutral Messaging Language
PL	Planning
RCS	Real-Time Control Systems
SP	Sensory Processing
VJ	Value Judgement
VRML	Virtual Reality Modeling Language
WM	World Modeling

DEFINING OPENNESS

A controller for manufacturing equipment may have varying degrees of openness. We are proposing an initial taxonomy of openness, described in terms of increasing openness in Table 1.

An Institute of Electrical and Electronic Engineers (IEEE) standards dictionary (IEEE, 84) defines the following terms as:

Computer component: any part, assembly, or subdivision of a computer

Software component: a basic part of a system or program

Subsystem: a division of a system that in itself has the characteristics of a system

Software subsystem: a group of assemblies or components or both combined to perform a single function.

Table 1: A Taxonomy of Controller Openness

Type of Openness	Description	Data or Software Openness	Software Rebuild Required
Output Data Openness	making data available at intermediate points within the control loops (e.g., providing the raw measured points so that custom analysis packages may be applied to assess the quality of the part)	Data	No
Input Data Openness	accepting data at certain points within the control loops (e.g., providing user control of number of measure points that are generated for certain features, altering the velocity of non-measurement moves)	Data	No
Limited Component Openness	having a limited set of coarse-grained replaceable components or interfaces specified (e.g., allowing for custom operator interfaces to be integrated)	Software	Yes
Subsystem Openness	allowing subsystems to be swapped (e.g., replacement of servo nodes with different algorithms)	Software	Yes
Algorithm & Component Openness	allowing access below the control node resolution (e.g., replacing a trajectory planner within the behavior generation portion of a control node)	Software	Yes
Plug and Play	allowing plug and play capabilities, where a piece of hardware can be swapped and the system software recognizes the change and adapts without having to rebuild the system	Software	No

We further define the following:

Node or controller node: a software unit capable of performing intelligent behavior based on the node's intelligent perception process, as specified in the Real-Time Control System (RCS) reference model architecture, described in detail in a later section. A node, therefore, typically encompasses multiple software components

A FRAMEWORK FOR OPENNESS

Several factors facilitate a system's openness. Assumptions regarding which hardware, operating system, or other infrastructure and which versions thereof must be known. The major functions of the system must be well defined. The "boundaries" of the components, sub-systems, and other modules must be clearly defined. Their interfaces must be unambiguously specified, but just as importantly, the behavior and constraints of these sub-systems must be clarified. These definitions are critical in order to ensure that vendors provide subsystems that function correctly with the rest of the system. Ideally, the component and interfaces are based on standards.

A framework describing the controller – its software pieces and their interfaces to hardware – is necessary in order to communicate the required definitions to the builders or enhancers of the various pieces. In the Intelligent Systems Division, we have been developing a reference architecture that can serve as such a framework (Albus, 94; Huang et al., 95). The RCS Architecture provides a building block approach for construction of complex systems. RCS specifies the decomposition of the system according to functionality and physical components along one dimension and into appropriate levels within a hierarchy along another dimension. The Intelligent System Architecture for Manufacturing (ISAM) is a version of RCS for advanced manufacturing systems. In ISAM, the basic building block is a control node, which contains the essential elements necessary for intelligent control: sensory processing (SP), behavior generation (BG), world modeling (WM), and value judgement (VJ). Associated with WM is a Knowledge DataBase (KB), which contains longer-term information. A graphical depiction of an ISAM node is shown in Fig. 1. Each node receives goals from its superior and, through the orchestration of BG, WM, VJ, and SP, generates a finer resolution set of goals for its subordinate nodes. The RCS control node uses an estimated model of the world, generated via SP and WM, to assess its progress with respect to the goals it was given and to make necessary adjustments to its behavior. BG is further broken out into sub-modules (not shown in Fig. 1), including a planning portion (PL) and a set of executors (EX), one for each subordinate. PL generates plans for each subordinate, while the corresponding EX executes the plan, coordinating actions between subordinates, and correcting for errors between the plan and the state of the world estimated by WM.

The control nodes are assembled into an architecture, guided by the RCS/ISAM methodology, which is designed to control complexity of the individual levels within the hierarchy. The decomposition for manufacturing systems is according to the following levels: shop, cell, workstation, task, elemental move (emove), primitive (prim), and servo. This

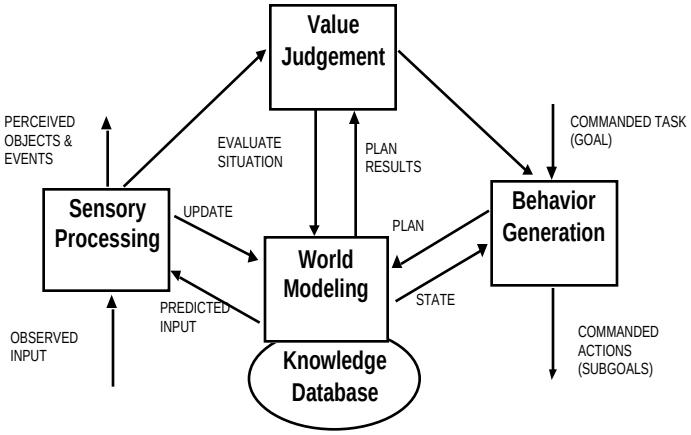

Figure 1: An ISAM Control Node

hierarchy covers the shop floor organization, where work cells have several pieces of equipment, and a workstation has several subsystems (e.g., a machine with a tool magazine and tool changer). The task level corresponds to a subsystem within the individual workstation, and the levels below task accomplish the movements and other actions of the individual subsystem. Only at the lowest level, servo, does the software interact with hardware. The actuators and sensors interface with this level. This helps to isolate the interfaces to specific drives or sensors.

Several commercial and research systems have been built based on RCS (Albus, 95). These include coal mining automation (Horst, 93), the NBS/NASA Standard Reference Model Architecture for the Space Station Telerobotic Servicer (NASREM) (Albus et al., 89), and a control system for a U.S. Postal Service Automated Stamp Distribution Center (USPS, 91). Manufacturing applications include the Open Architecture Enhanced Machine Controller (Albus and Lumia, 94; Proctor et al., 93), a welding cell (Rippey and Falco, 97), and several systems in the National Advanced Manufacturing Testbed (Luce et al., 99).

AN ARCHITECTURE FOR INSPECTION

Figure 2 shows the architectural decomposition for an inspection system. This is the architecture that is used in our testbed. Within that testbed, a CMM and associated devices are integrated into a cell that communicates with the rest of the enterprise. The level above the cell (shop) or an operator sends a command to initiate an inspection of a part. The cell notifies its superior when it completes the inspection and if there were any problems. The coordinate measurement machine is a Sheffield Cordax[1] three axis cartesian arm. A touch probe and a charged-couple device (CCD) camera are affixed to the end of the arm. Our system also incorporates a feature-based inspection planner and the ability to do fixtureless inspection.

[1] **Disclaimer:** Certain commercial equipment, instruments, or materials are identified in this article to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by NIST, nor does it imply the materials or equipment identified are necessarily the best for the purpose.

The cell level receives a command to inspect a previously made part and makes a cell level process plan. The planning process converts the goal of inspecting the part into a set of tasks to inspect segments of the part based on setup requirements. A setup means a particular placement or fixturing of the part on the inspection table. A prismatic part often requires multiple setups so that the multiple facets can be inspected. The knowledge requirement to facilitate the planning includes the descriptions for the part, the features on the part, the optional fixture, the plan format, and the setup. The generated plan is executed and the resulting commands are given to the workstation controller, one setup a time, to make the corresponding plans.

The workstation controller performs two major functions: inspection planning and inspection coordination. First, the workstation controller generates inspection process plans for a setup, as commanded by the cell controller. The process plan includes a series of inspection tasks corresponding to the features associated with the setup. The operator can specify which features to inspect and how thoroughly they should be inspected.

Second, the workstation controller coordinates the execution of an inspection operation. We have developed a vocabulary to describe the coordination plan steps. The steps include loading and unloading a part, using the vision subsystem to estimate the pose of the part, and running the inspection programs for a given setup. Operators use a text editor to generate the plan and the controller is equipped with an interpreter to interpret the plans.

This testbed is capable of performing both fixtured and fixtureless inspection. In the fixtureless inspections, once the part is placed, the vision controller subordinate is commanded to perform a perception process to estimate the part pose. Finally, the workstation executes the generated inspection process plans. Two subordinates, the Coordinate Measuring Machine (CMM) and fixturing, receive and execute the workstation output commands.

The CMM task controller performs two major functions. First, it generates inspection programs, written in the Dimensional Measuring Interface Standard (DMIS) language, for a feature as command by the workstation level. Second, the DMIS interpreter translates the DMIS statements into command language native to the inspection controller. The executor sends these messages to the three emove subordinates, for motion control of the CMM, inspection tool control, and vision-based part pose estimation. The task controller also receives inspection data, which the interpreter uses to compute and evaluate the inspection results.

The fixturing controller performs the fixturing or placement of the part according to the setup specification. Control system developers provide the placement or fixturing instructions, displayed via a web browser, to facilitate remote operations of the node. The execution is performed by an operator.

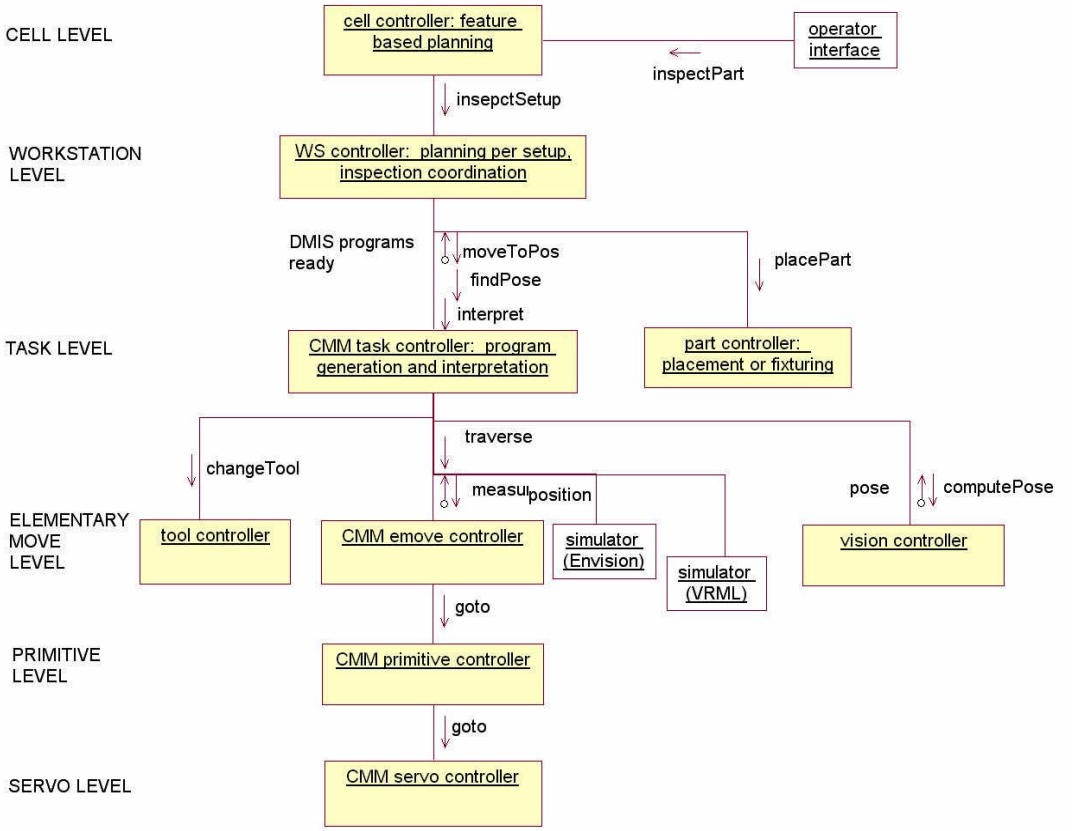

Figure 2: The Inspection Workstation Control Hierarchy

The CMM emove controller receives the interpreted motion commands, traverse or measure, and moves the CMM accordingly. The speeds and accelerations must be specified, either by the task controller as a part of the command parameters, or by the emove controller. In the current implementation, we use the latter. When traverse is done in an open space, the controller uses a high speed. The measure commands use a low speed.

The tool controller receives commands to change the probe. The probe type and installation instructions are displayed on a web page. The execution is performed by an operator, who clicks a done or error button on the web page to indicate completion. An error signal halts the entire operation.

The part pose estimation subsystem uses vision to determine position and orientation of the part to facilitate the coordinate system transformation among the different levels.

The primitive and servo levels deal with the trajectory generation and servoing of the CMM arm according to input commands.

All the nodes execute initialization tasks at the beginning of an operation to verify the interfaces and initialize certain variables. Some of the nodes execute exit, close, halt, or stop tasks to manage the stopping and restarting of operations.

OPENNESS ASPECTS OF THE INSPECTION ARCHITECTURE

The inspection testbed we have built demonstrates openness at a variety of levels. The nodes and their interfaces are clearly defined, enabling future development of plug and play. Precise definitions of some of the software components internal to control nodes also enable plug and play. We have also been experimenting with different types of open-system integration approaches. One of them uses C++ code templates, called the Generic Shell to build skeleton control hierarchies for applications with the capability to easily swap in and out software components. Another is based on a commercial tool, called ControlShell, by Real-Time Innovations, Inc., that offers a similar capability. Intermediate plans and data are accessible throughout the hierarchy. We will describe briefly the open aspects of the testbed and how they can be utilized.

The testbed was built using the Neutral Messaging Language (NML), which enables definition of messages that can be sent across a network. NML is built on top of the Communications Management System (CMS), which provides uniform access to buffers to multiple reader or writer processes on the same processor, across a backplane, or over a network. All communications occurring between nodes use NML, hence their command and status are readily available for diagnosis and other uses. A diagnostic tool, which was developed within ISD, graphically displays the hierarchy's status and messages. If a problem develops in the functioning of the system, its location can be quickly traced by looking at the diagnostic tool. Figures 3 and 4 show screen shots from the diagnostic tool. Figure 3 shows a different perspective from Fig. 2. In Fig. 3, the control nodes are broken out into two computing processes, planning and execution.

The commands between nodes can also be captured and used for animations and simulations. Our testbed included two such tools, as shown in Fig. 2. A simulation of the Cordax, built using Envision package (by Deneb), was hooked into the hierarchy via NML. It listened to the commands that were sent to the Emove level from the Task node. These "goto" commands were read by the Envision program, which used the kinematic model of the Cordax to compute joint motions to achieve the desired positions. Inspection programs can be validated at a high level using this simulation instead of using the Cordax, which is a more expensive resource. The second visualization tool was built using Virtual Reality Modeling Language (VRML). A model of the Cordax geometry was defined in VRML. The VRML display listened to the joint position feedback messages coming from Emove into Task and used those to drive the motion of the model, which could be displayed on any system with a VRML viewer. This tool can be used to validate motion algorithms at a lower level.

The process planning system is the Feature-Based Inspection and Control System (FBICS). FBICS provides human-readable files containing the inspection plans at the cell, workstation, and task levels. These files' contents are all defined by EXPRESS schemas. The cell level produces plans for each of the setups required to inspect the part. The workstation level produces plans for inspecting the desired features. The task level planner creates the DMIS inspection code. Users also have access to files that define the behavior of the FBICS system. For example, the shop_options file lets users select which features to inspect. They can specify that the system inspect all features, inspect no features, inspect any feature with a parameter having a tolerance, inspect any feature which has any parameter with a "tight" tolerance, or let the user decide for each feature. The definition of what is a "tight" tolerance is another user option.

To facilitate openness, the testbed utilizes standards wherever possible. FBICS uses various International Organization for Standardization (ISO) standards. It uses EXPRESS as its modeling language. STEP AP 224, which specifies parametric machining features, is used for defining features to be inspected as well as the overall part geometry. STEP AP 203 is used to define fixturing. The inspection programs are represented by the Dimensional Measuring Interface Standard (DMIS).

The system was implemented in a distributed manner. Several different computers and operating systems were used in the testbed. We used a Power PC/VxWorks target for real-time control, a Sun Sparc/Solaris target for the vision subsystem, and two more separate Sparc/Solaris platforms for inspection planning and plan execution. The Envision simulation ran on an SGI/Irix workstation. The VRML simulation was run from a PC with Windows NT. Each node ran as one or more separate processes. The organization of the overall architecture in this multi-process and distributed manner facilitates openness. Once the cost of separating out modules into different processes and debugging their interfaces has been paid, other users of the same architecture can leverage the advantages of modularity.

Subsystem replacement was used in several parts of the testbed. Primarily, this was done for swapping a simulated subsystem for the real one that ran the hardware. For example, the developer testing the higher levels of the hierarchy (Cell through task) could substitute the real CMM or Vision branches with simulated versions. Similar substitutions were performed regularly at lower levels, such as the Prim and Servo level.

Intra-node openness is also present in the inspection testbed. The FBICS planners have clearly defined application programming interfaces (APIs). These can be used in other testbeds or can be replaced with different underlying algorithms. A great deal of effort was spent investigating component specifications under support of another project. The part pose estimation algorithm, which resides within the vision controller branch of the hierarchy, was used to study component specifications to facilitate component-based software development (Horst et al., 97). The specifications of this component are extremely detailed and are an effort in understanding how to represent semantic as well as syntactical information about a piece of software.

The testbed does not yet demonstrate plug and play openness. Having exercised the various other types of openness which are prerequisites to plug and play, we are confident that it will be achievable. A plug and play application of interest to inspection system users is the ability to swap sensors and have the system identify the type of sensor and automatically update the parameters for planning its use.

FUTURE DIRECTION

Now that we have completed the basic implementation of a testbed for open architectures for inspection, we are planning to undertake further experiments in validating the openness. We would like to collaborate with industry to validate whether the architecture is a reasonable one from their perspective and to conduct joint experiments to verify the different types of openness.

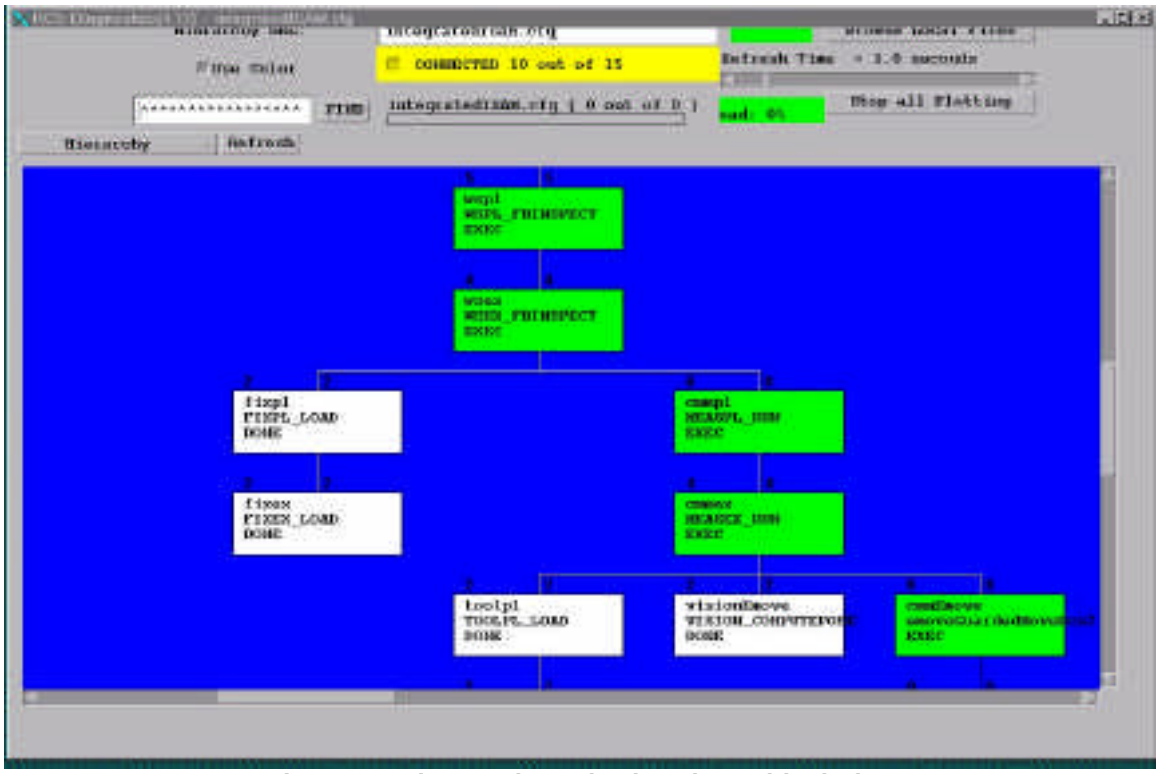

Figure 3: Diagnostic tool—the Hierarchical View

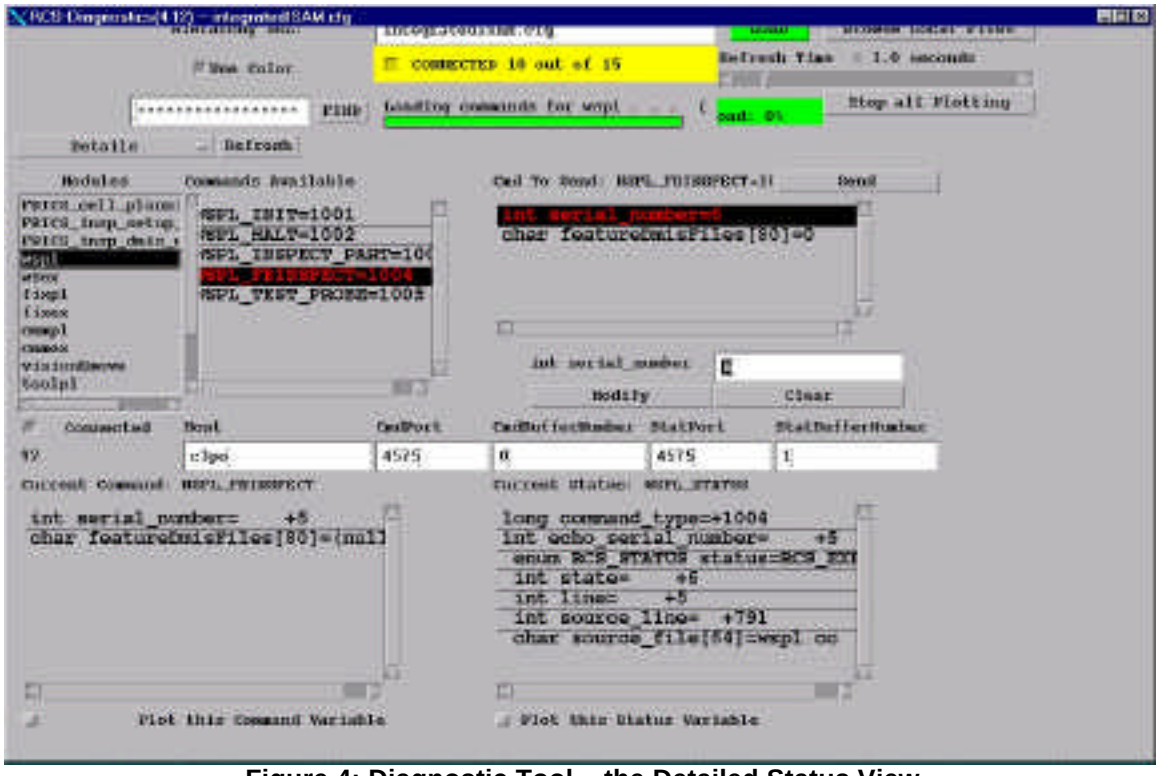

Figure 4: Diagnostic Tool – the Detailed Status View

CONCLUSION

We have described a testbed for experimenting with open architectures for inspection systems. We have proposed a taxonomy of openness for inspection controllers, which defines different degrees of openness. Our testbed has demonstrated both data and software openness. Intermediate planning results and other data are accessible through clearly defined interfaces or data files. Users can input control parameters to the system through interfaces or data files. Software subsystems and components within those subsystems may be replaced because of clearly defined module interfaces. These various kinds of openness are prerequisites for plug and play – where a piece of hardware can be dynamically swapped without requiring a rebuild of the system's software. We expect to demonstrate plug and play within the inspection testbed. Our intention is to collaborate with industry to study open architectures for inspection and to jointly promote their benefits.

REFERENCES

Albus, J.S., Lumia, R., Fiala, J., and Wavering, A. 1989. NASREM – The NASA/NBS Standard Reference Model for Telerobot Control System Architecture. Proc. of the 20th International Symposium on Industrial Robots, Tokyo, Japan.

Albus, J.S., 1994, A Reference Model Architecture for Intelligent Systems Design, NISTIR 5502, National Institute of Standards and Technology, Gaithersburg, MD.

Albus, J.S., Lumia, R., 1994. The Enhanced Machine Controller (EMC): An Open Architecture Controller for Machine Tools. Journal of Manufacturing Review, Vol. 7, No. 3, pgs. 278-280.

Albus, J. S., 1995. The NIST Real-time Control System (RCS): An Application Survey. Proc. of the AAAI 1995 Spring Symposium Series, Stanford University, Menlo Park, CA.

Horst, J. A., 1993. Coal Extraction Using RCS. Proc. of the 8th IEEE International Symposium on Intelligent Control, Chicago, IL, pp. 207-212.

Horst, J. A., Messina, E., Kramer, T., Huang, H. M., 1997. Precise Definition of Software Component Specifications. Proc. of the 7th Symposium on Computer-Aided Control System Design (CACSD '97), Gent, Belgium, pp.145-150.

Huang, H., Michaloski, J., Tarnoff, N., and Nashman, M., 1995. An Open Architecture Based Framework for Automation and Intelligent System Control, Invited Paper for the IEEE Industrial Automation and Control Conference 95,, Taipei, Taiwan, May 1995.

IEEE Standard Dictionary of Electrical and Electronic Terms, ANSI-IEEE Std. 100-1984, The Institute of Electrical and Electronic Engineers, New York, NY.

Luce, M. E. Stieren, D. C. , Densock, R. J., 1999. National Advanced Manufacturing Testbed, NISTIR 6383, National Institute of Standards and Technology, Gaithersburg, MD.

Proctor, F., et al, Open Architectures for Machine Control. 1993. NISTIR 5307, National Institute of Standards and Technology, Gaithersburg, MD.

Rippey, W., Falco, J., 1997, The NIST Automated Arc Welding Testbed, Proceedings of the 7th International Conference on Computer Technology in Welding, San Francisco, CA.

United States Postal Service, 1991, Stamp Distribution Network, Advanced Technology & Research Corporation, Burtonsville, MD. USPS Contract Number 104230-91-C-3127 Final Report.